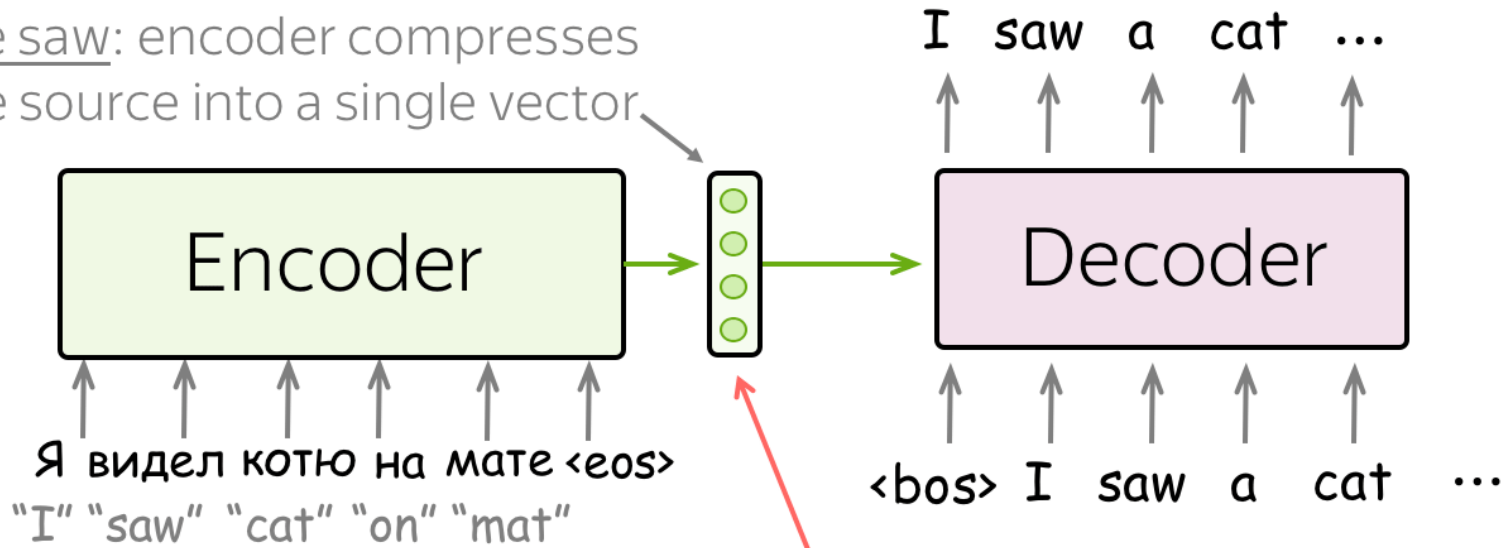

Foundations for Natural Language Processing

Transformer (part 1)

Ivan Titov
(with graphics/materials from Elena Voita)

Recap: Vanilla encoder-decoders

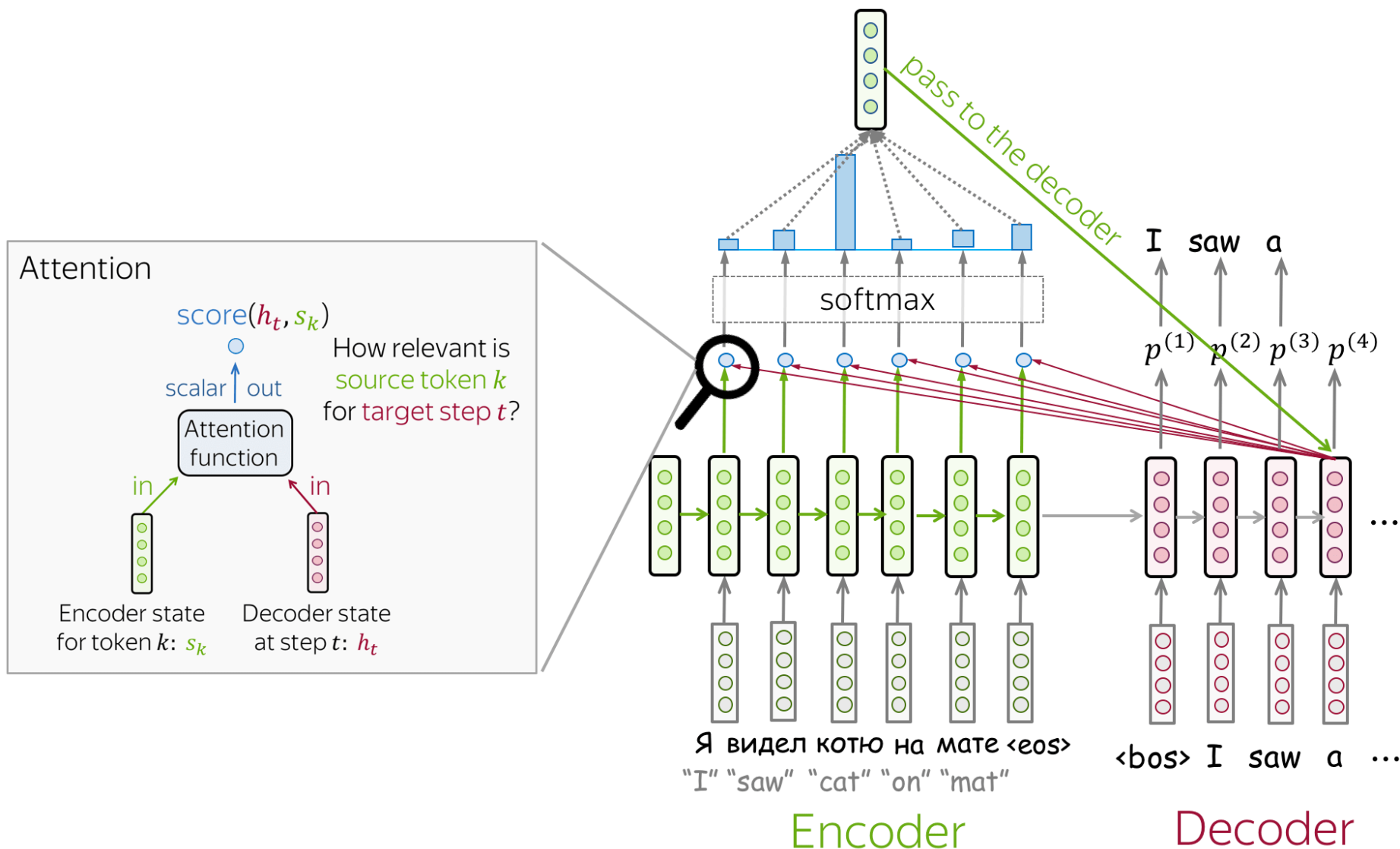
We saw: encoder compresses the source into a single vector



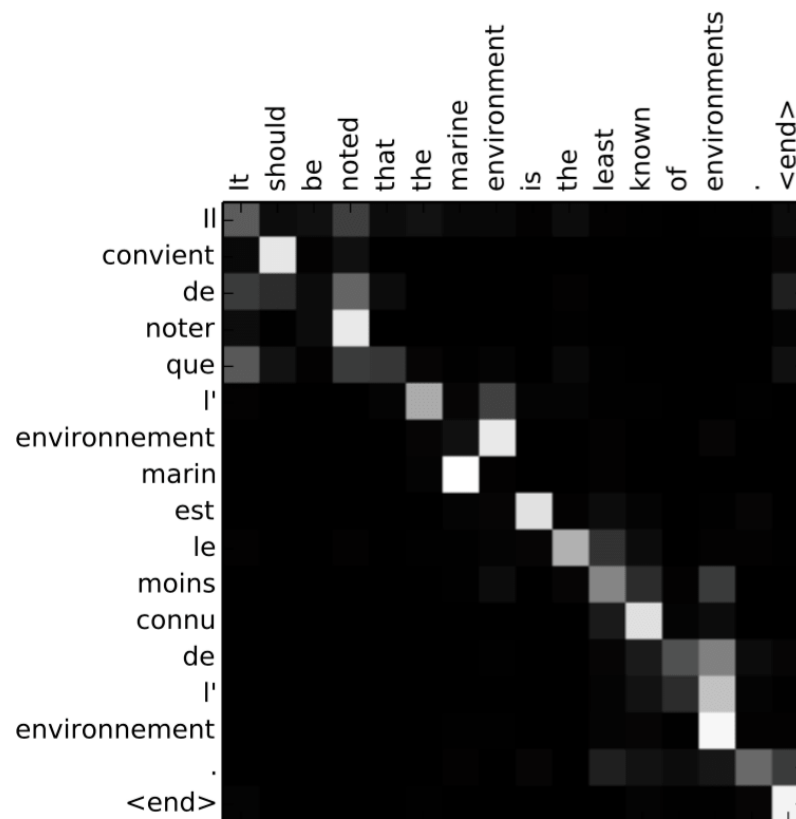
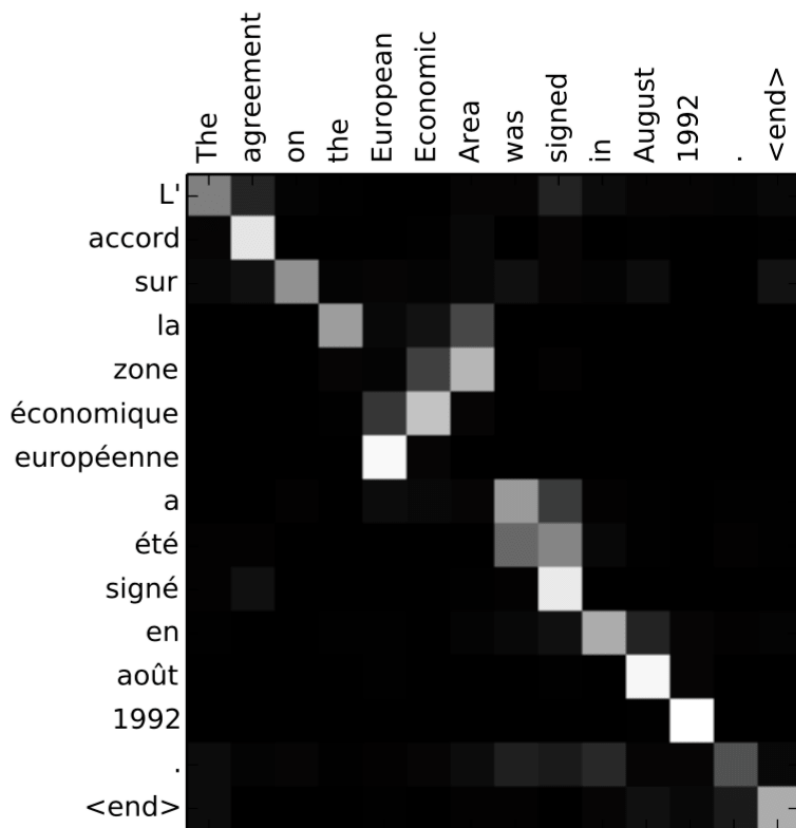
Problem: this is a bottleneck!

Recap: Attention

Model learns to 'pay attention' to most relevant source tokens



Recap: attention is (a bit like) alignment



Transformer

“Attention is all you need”

	Seq2seq without attention	Seq2seq with attention	Transformer
processing within encoder	RNN/CNN	RNN/CNN	attention
processing within decoder	RNN/CNN	RNN/CNN	attention
decoder-encoder interaction	static fixed-sized vector	attention	attention

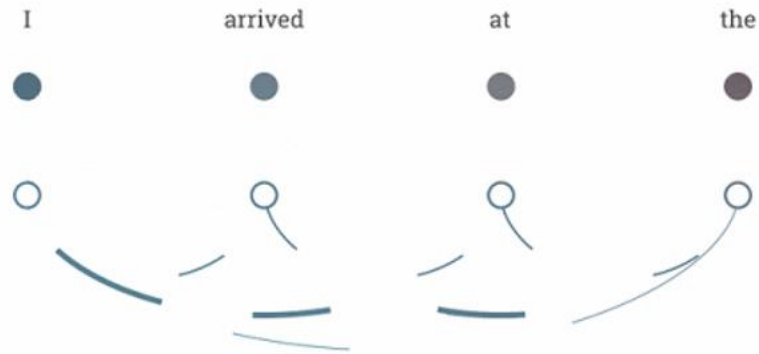
Transformer: Intuition

See animation here:

<https://blog.research.google/2017/08/transformer-novel-neural-network.html>

Transformer: Intuition

Encoding



See animation here:

<https://blog.research.google/2017/08/transformer-novel-neural-network.html>

Transformer: Intuition

Encoder

Who is doing:

- all source tokens

What they are doing:

- look at each other
 - update representations
- repeat
N times

Decoder

Who is doing:

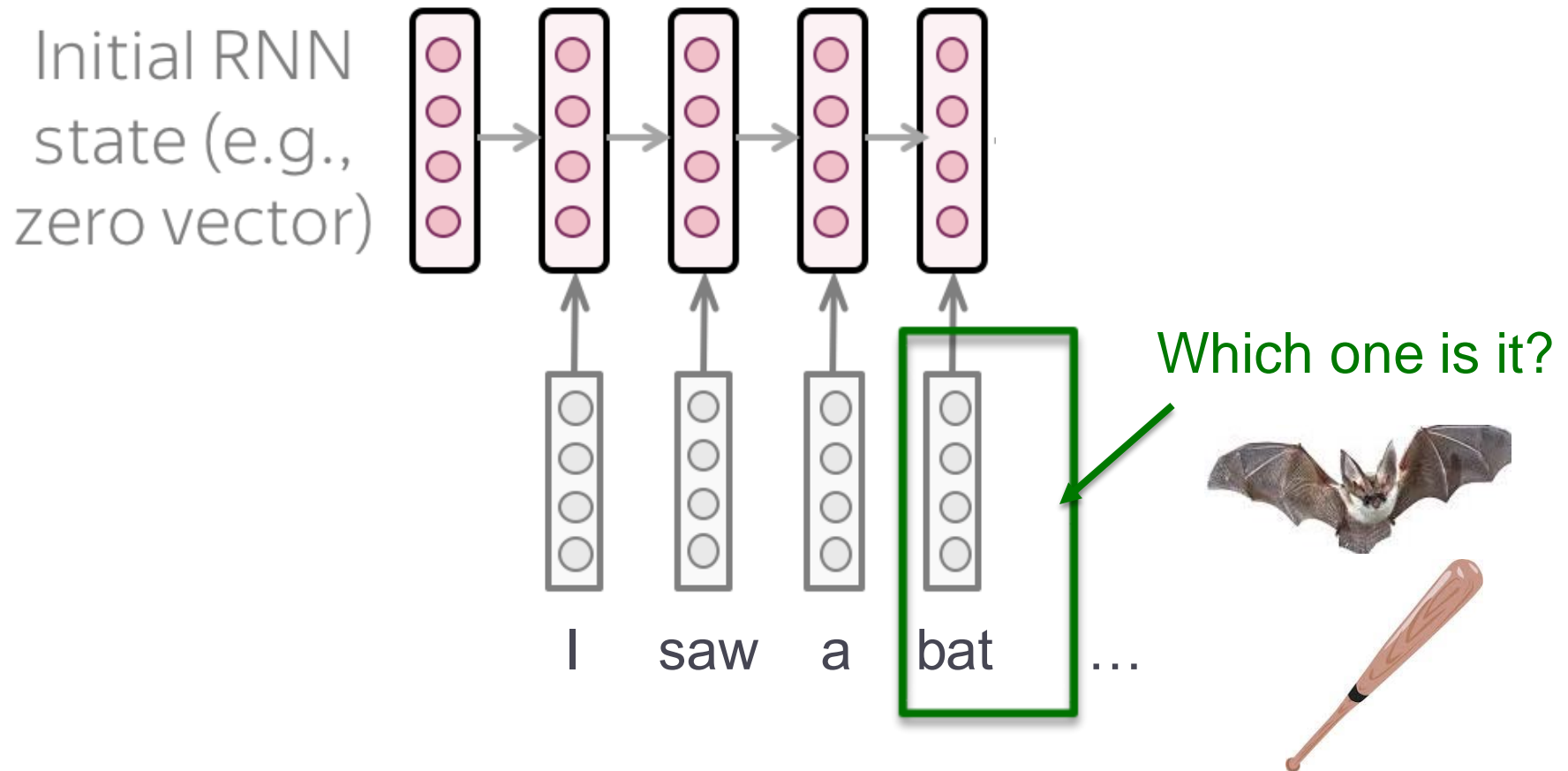
- target token at the current step

What they are doing:

- looks at previous target tokens
 - looks at source representations
 - update representation
- repeat
N times

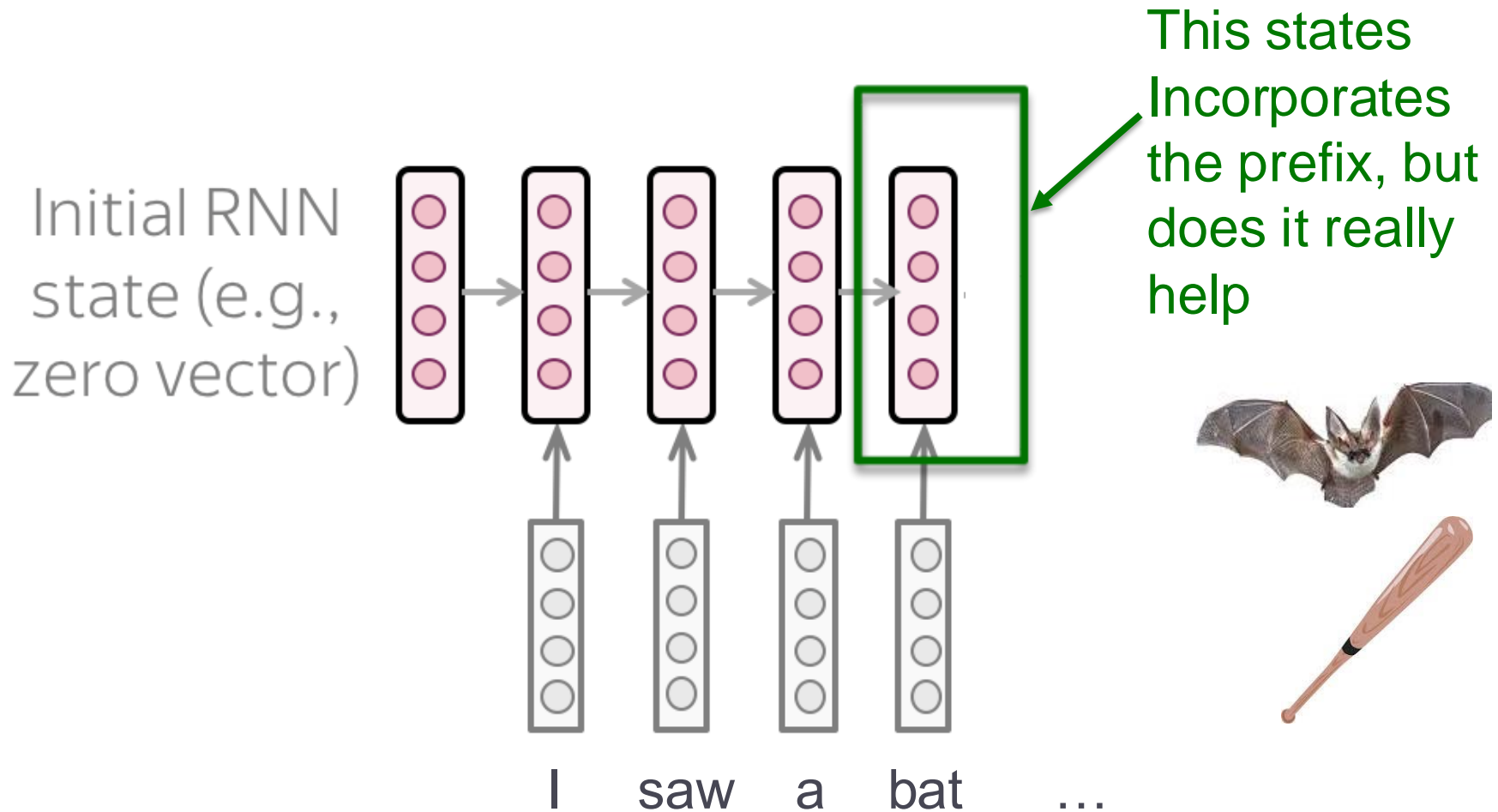
Why do we need self-attention in the encoder?

Let's recall RNNs



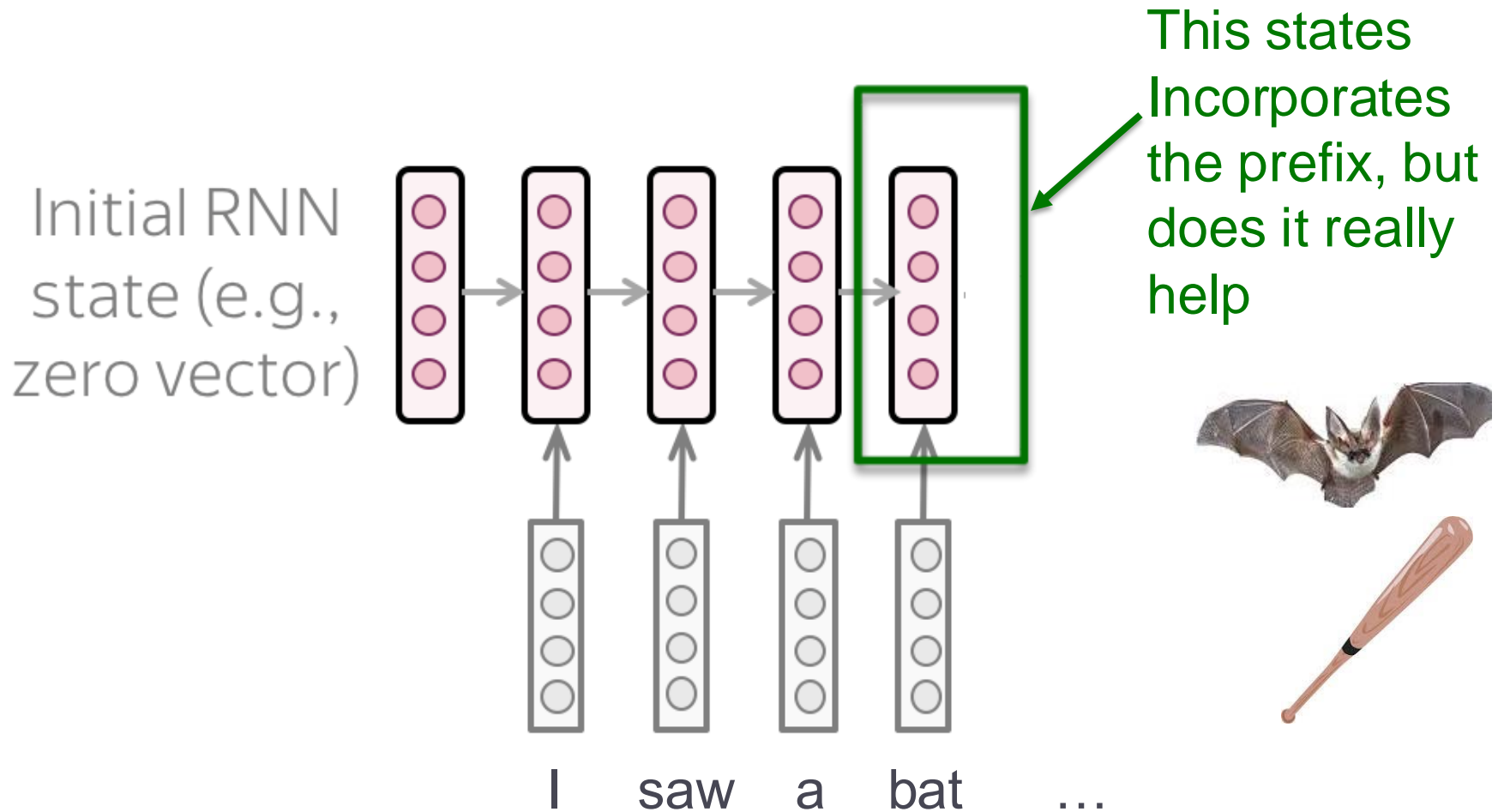
Why do we need self-attention in the encoder?

Let's recall RNNs



Why do we need self-attention in the encoder?

Let's recall RNNs



In RNNs we used bidirectional encoders to incorporate the information about the “future”

Why do we need self-attention in the encoder?

The self-attention is used in the Transformer to incorporate information about the context (including future context)



The process repeated multiple times, once per layer, iteratively refining the token representations

Learned end-to-end with an encoder-decoder model, so the model will learn to produce token representations useful for the decoder*

*this is going to be task-specific, e.g., representations useful for translation will be different from those for sentiment analysis or summarization

Encoder-decoder attention vs self-attention

Decoder-encoder attention is looking

- **from:** one current decoder state
- **at:** all encoder states

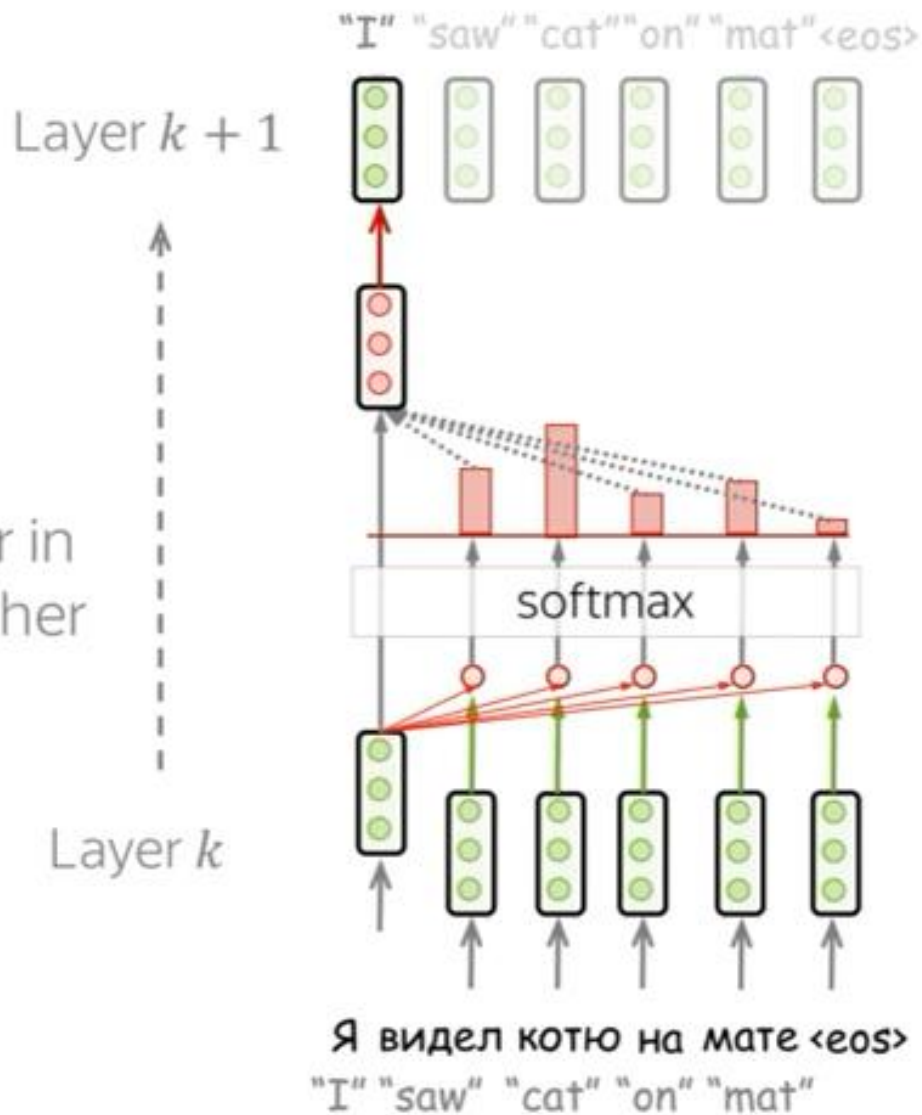
aka “cross-attention”



Self-attention is looking

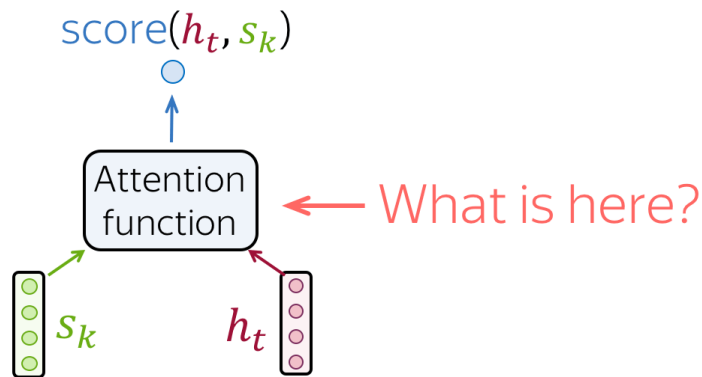
- **from:** each state from a set of states
- **at:** all other states in the same set

Self-attention



Tokens try to understand themselves better in context of each other

Recap: attention computation in encoder-decoder



Dot-product

$$h_t^T \times s_k$$

$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear

$$h_t^T \times W \times s_k$$

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

aka 'Luong attention'

Multi-Layer Perceptron

$$w_2^T \times \tanh \left[W_1 \times \begin{bmatrix} h_t \\ s_k \end{bmatrix} \right]$$

$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

aka 'Bahdanau' attention
(from the original paper)

Query-Key-Value attention

In self-attention, each token plays 3 different roles and has 3 different representations (1 per role)

1. query- asking for information;
2. key - saying that it has some information;
3. value - giving the information.

Query-Key-Value attention

They all 3 are a result of a linear transformation of the original representation

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

Query: vector **from** which the attention is looking

“Hey there, do you have this information?”

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

Key: vector **at** which the query looks to compute weights

“Hi, I have this information – give me a large weight!”

$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

Value: their weighted sum is attention output

“Here’s the information I have!”

Query-Key-Value attention

Each vector receives three representations ("roles")

$$\begin{bmatrix} W_Q \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

Query: vector **from** which the attention is looking

"Hey there, do you have this information?"

$$\begin{bmatrix} W_K \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

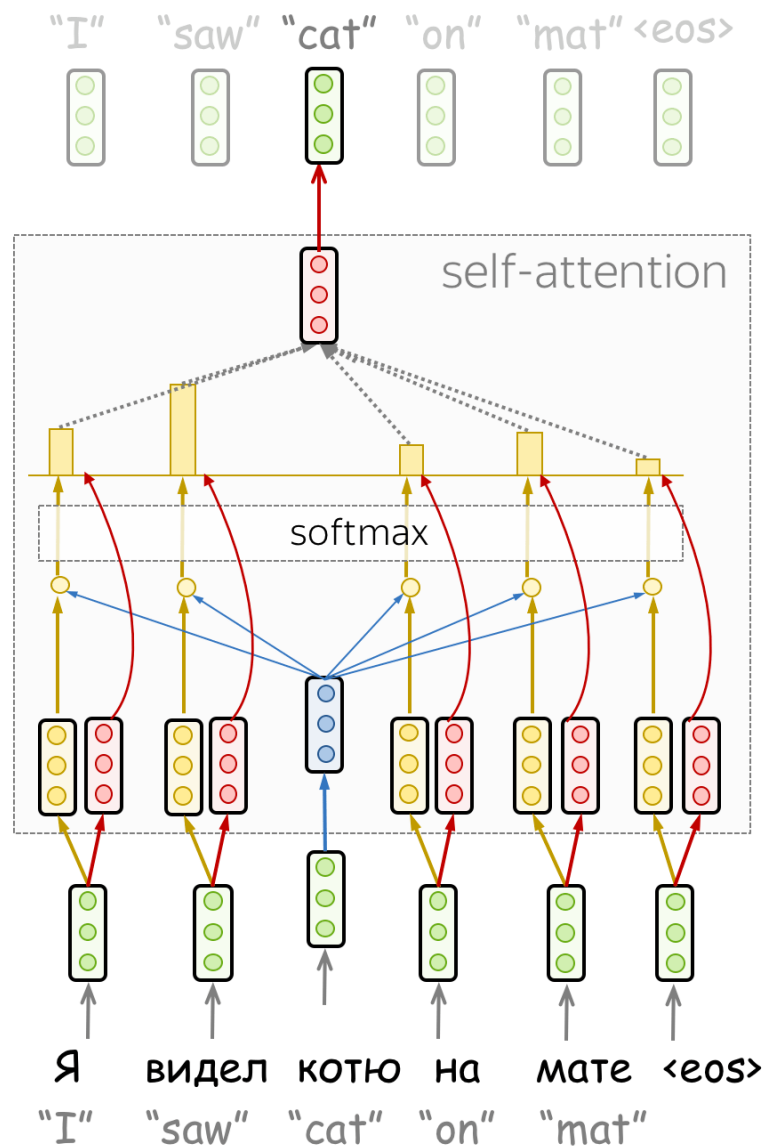
Key: vector **at** which the query looks to compute weights

"Hi, I have this information - give me a large weight!"

$$\begin{bmatrix} W_V \end{bmatrix} \times \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix} = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \end{bmatrix}$$

Value: their weighted sum is attention output

"Here's the information I have!"

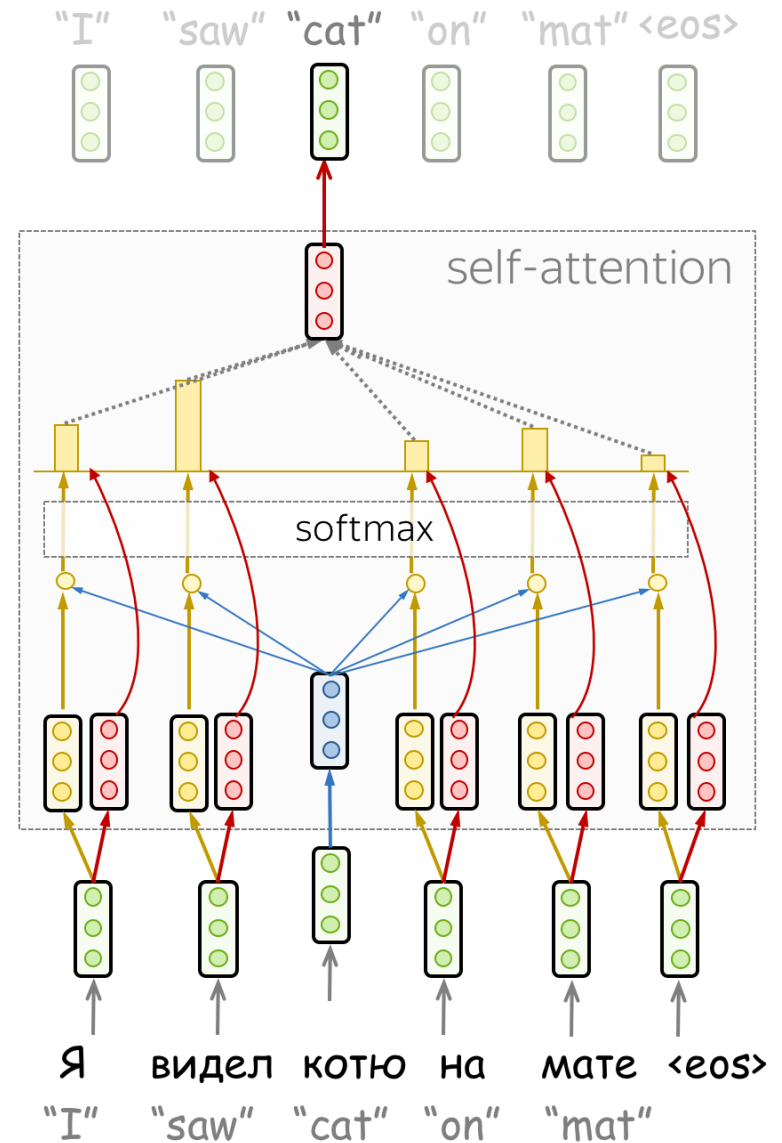


Query-Key-Value attention

$$\text{Attention}(q, k, v) = \frac{\text{Attention weights}}{\sqrt{d_k}} v$$

from q to k

vector dimensionality of K, V



QK parallelization

N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N

Transformer

“Attention is all you need”

processing
within **encoder**

processing
within **decoder**

decoder-encoder
interaction

Transformer

attention

attention

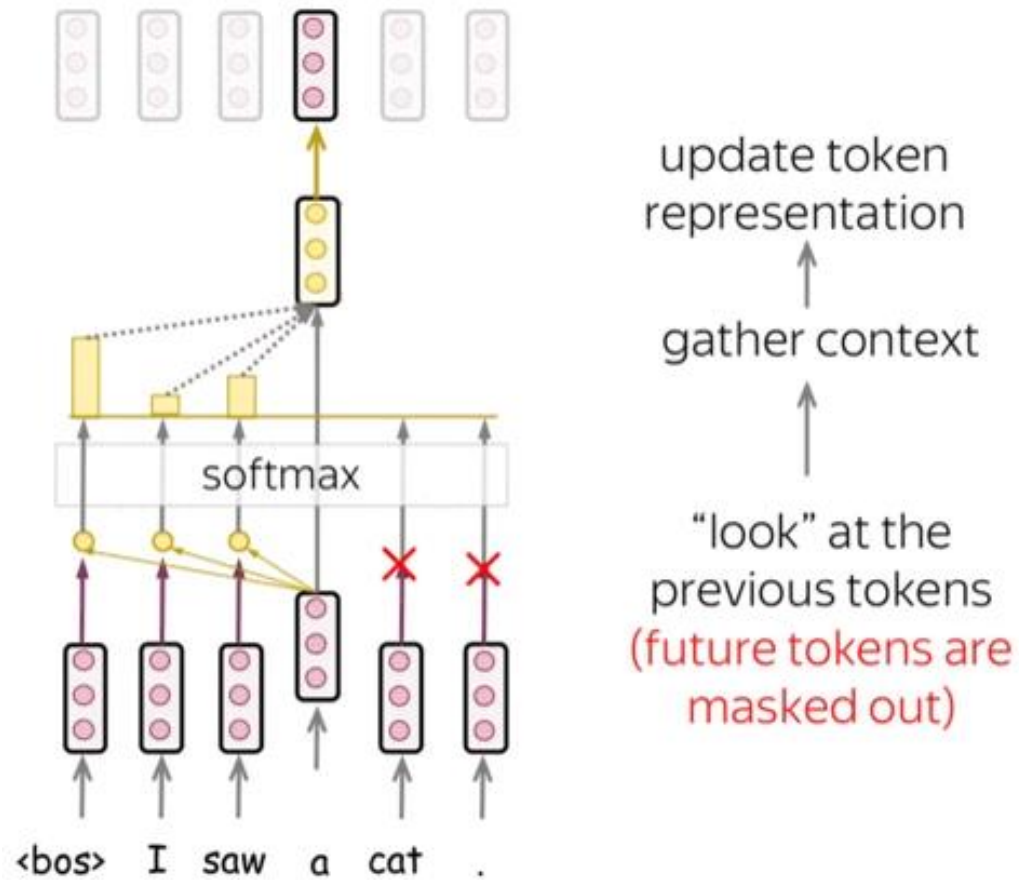
attention

Masked Attention

At inference time, the decoder does not have access to the future (because it has not generated it yet)

The future is known in training

But training needs to be **consistent** with inference, so we **'mask'** future tokens when training the representations in the decoder



(you can think of this as a trick enabling fast training)

QK parallelization with masking

N

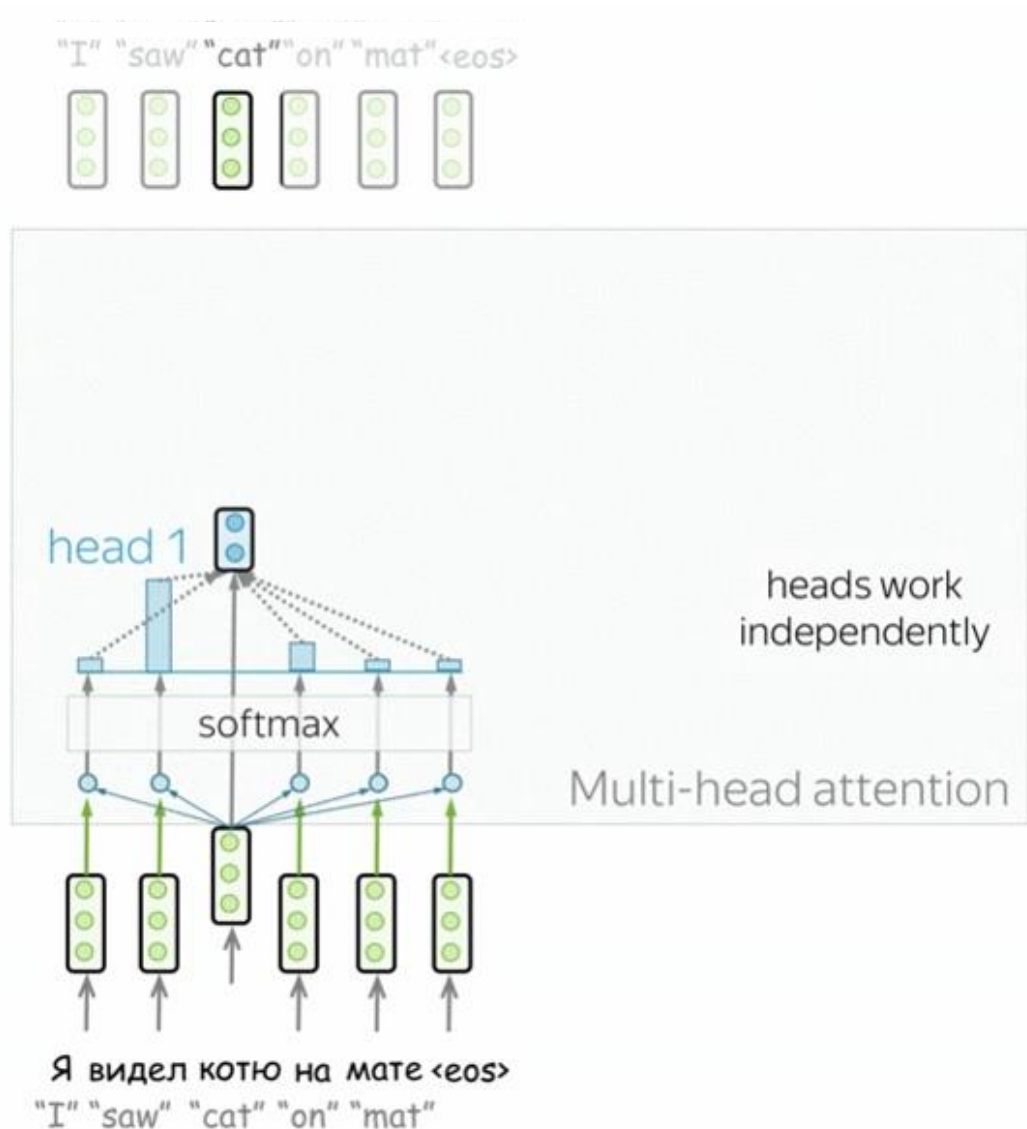
q1·k1	$-\infty$	$-\infty$	$-\infty$
q2·k1	q2·k2	$-\infty$	$-\infty$
q3·k1	q3·k2	q3·k3	$-\infty$
q4·k1	q4·k2	q4·k3	q4·k4

N

Multi-Head Attention

Each head specializes on a **different relation**

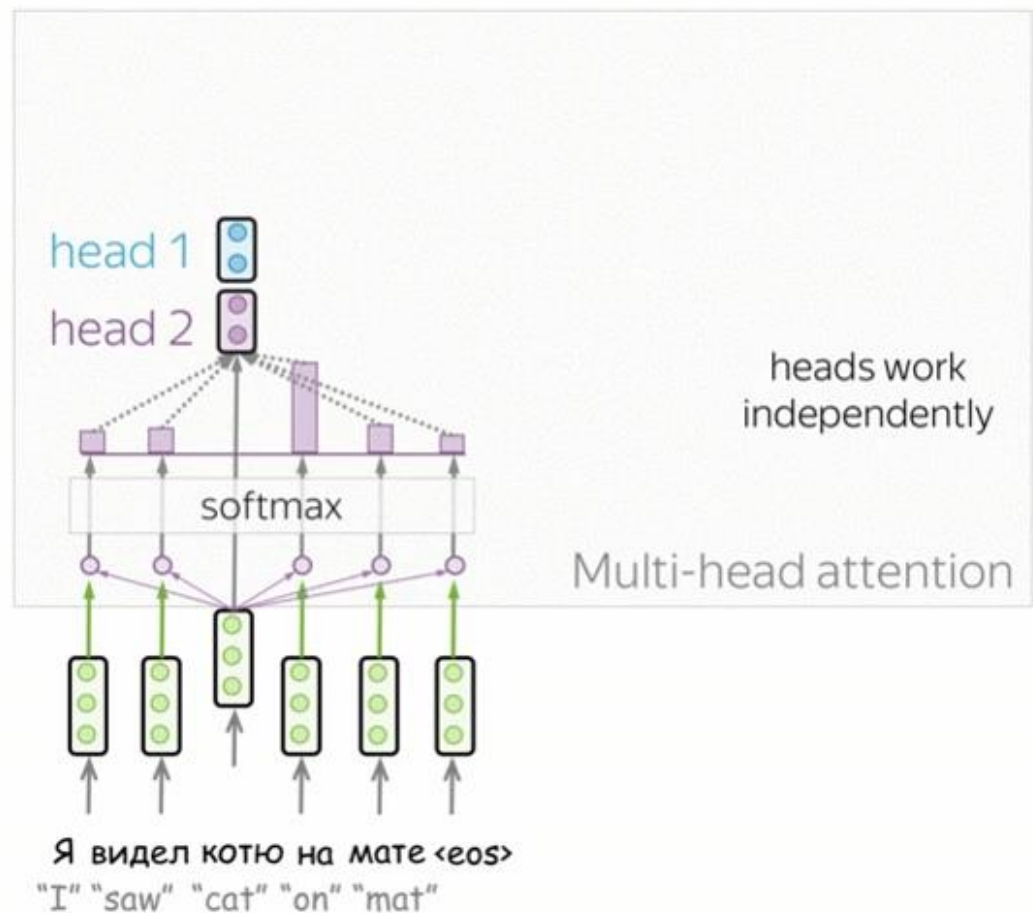
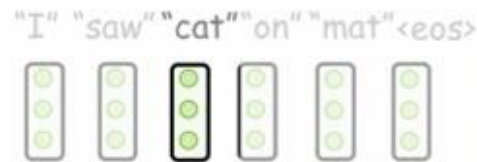
Intuition: there are **different relations between words** in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

Each head specializes on a **different relation**

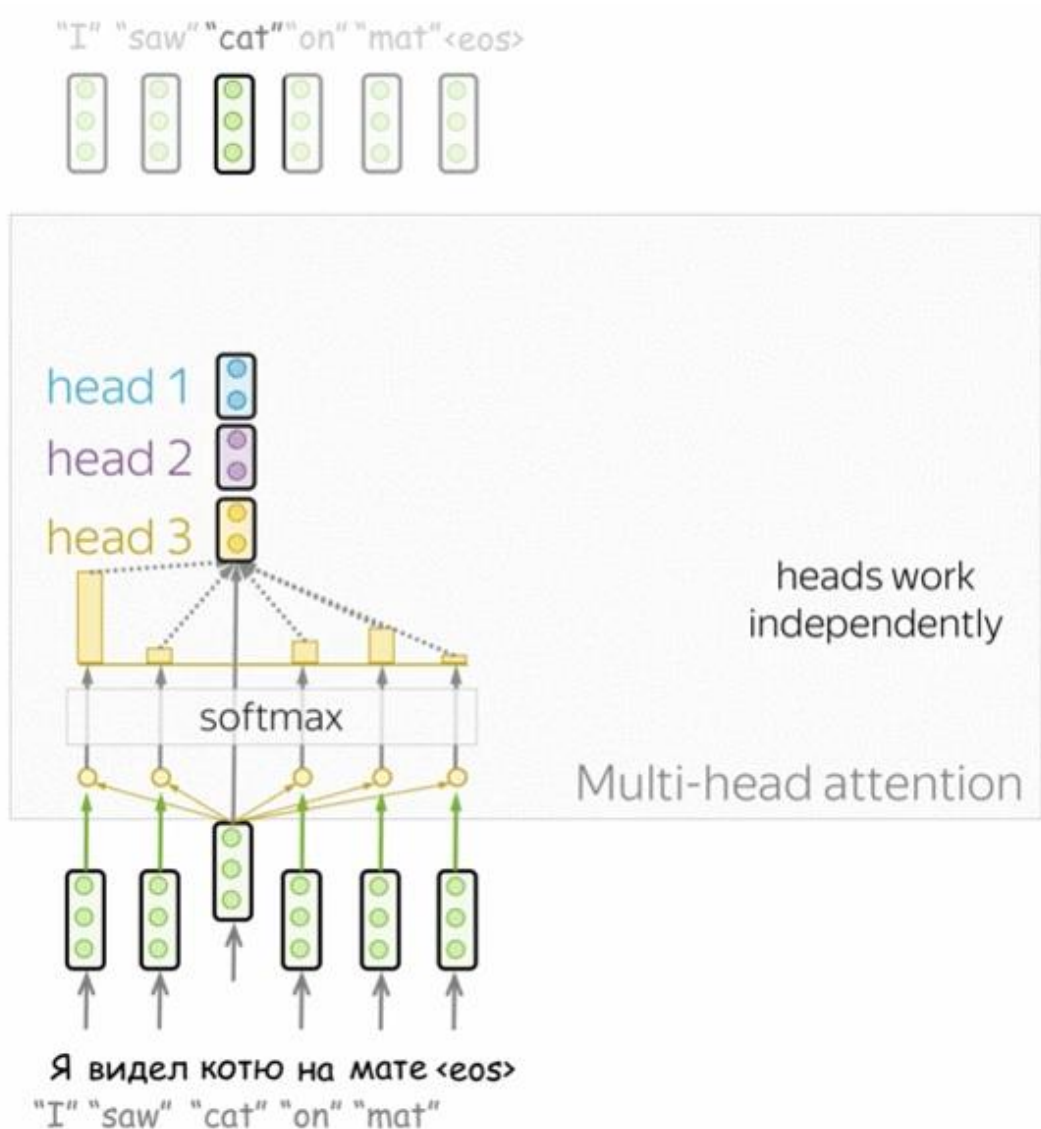
Intuition: there are **different relations between words** in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

Each head specializes on a **different relation**

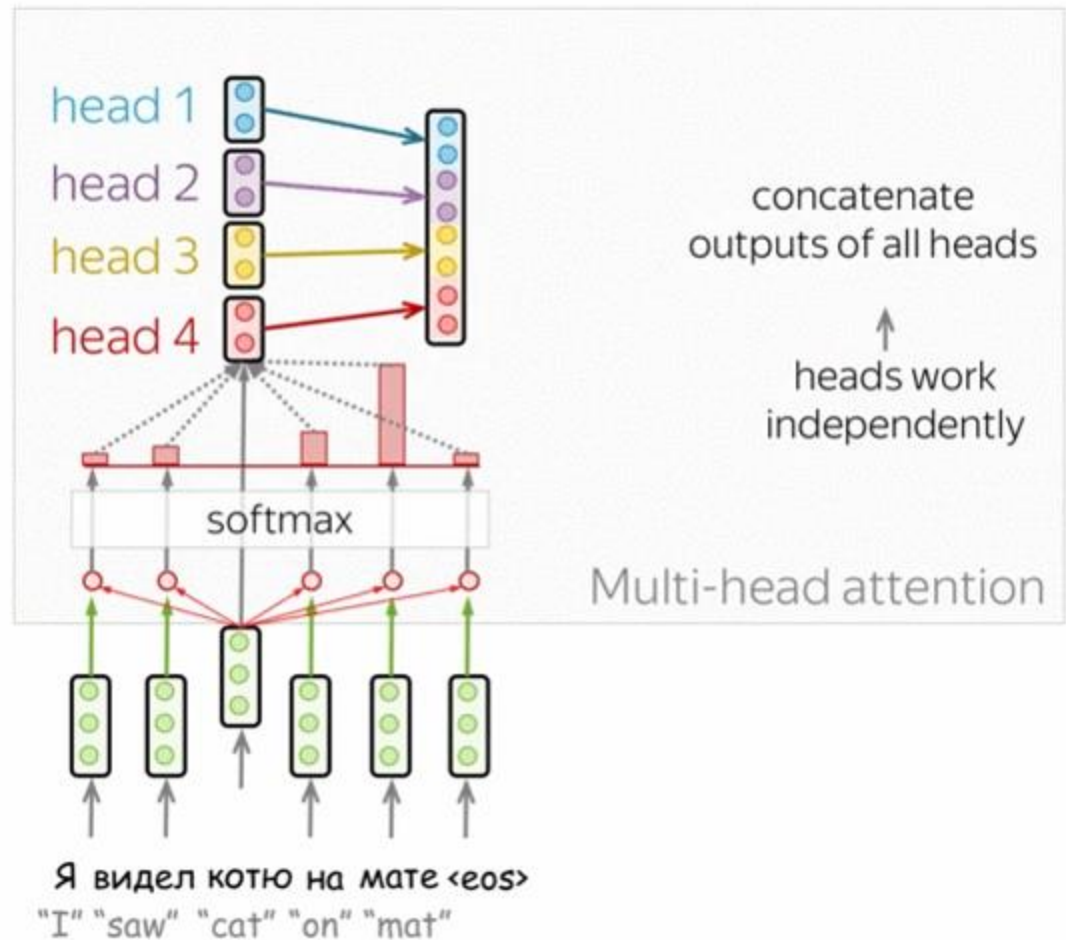
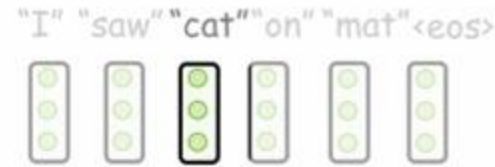
Intuition: there are **different relations between words** in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

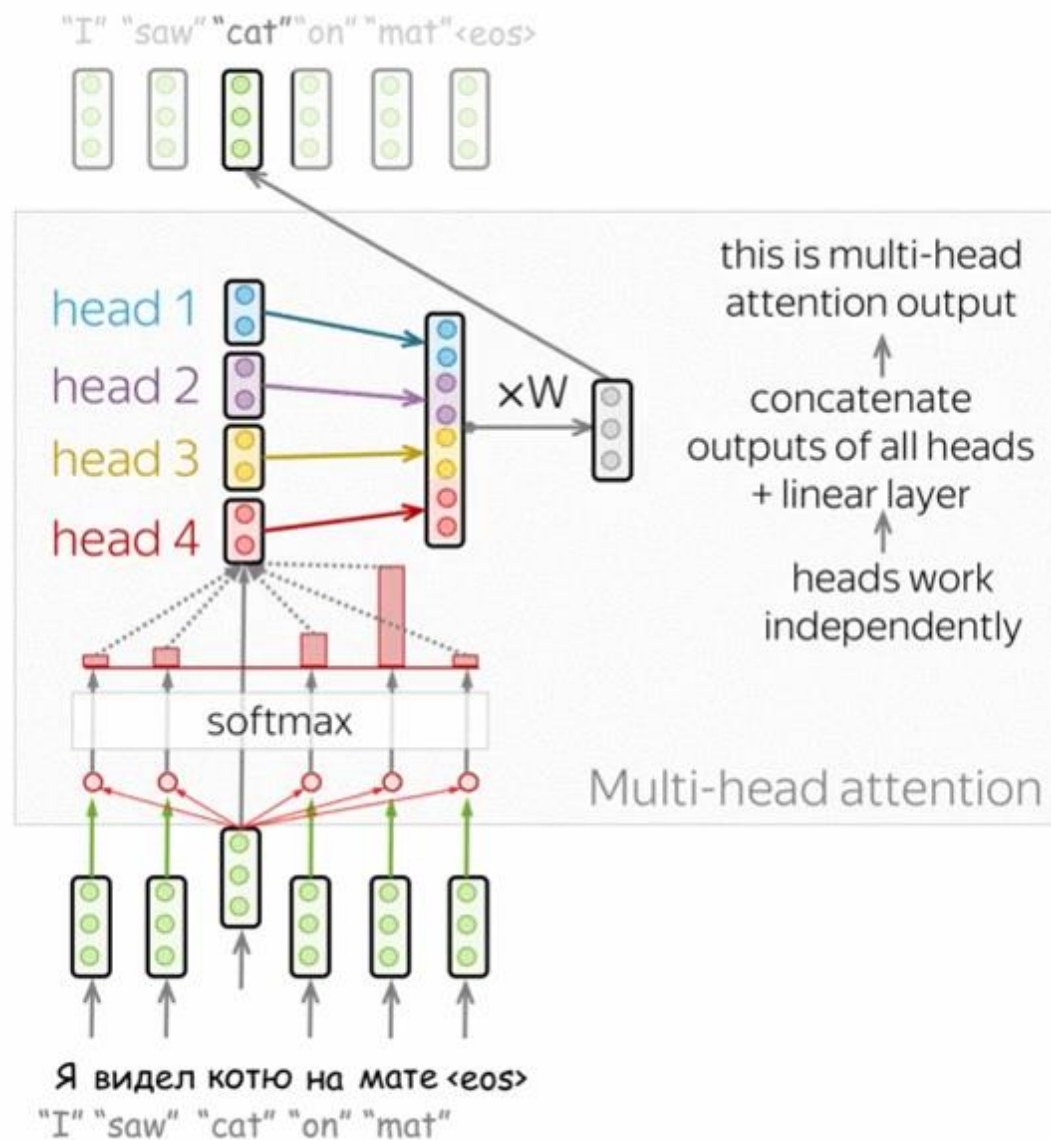
Each head specializes on a **different relation**

Intuition: there are **different relations between words** in a sentence (e.g., subject 'affects' a verb in a different way than its object)



Multi-Head Attention

Each heads performs independent QKV attention (with their own head-specific parameters, i.e. W_k , W_q , W_v matrices)



Multi-Head Attention

Let's say Q – is the set of states we look **from**

K - is the set of states we look **at**

V - is the set of states we **take values from** (usually $K=V$)

In encoder self-attention $Q=K=V$; *they are represented as matrices (states = rows)*

Multi-Head Attention

Let's say Q – is the set of states we look **from**

K - is the set of states we look **at**

V - is the set of states we **take values from** (usually $K=V$)

In encoder self-attention $Q=K=V$; *they are represented as matrices (states = rows)*

Each head has its own head-specific parameters, i.e. W_k ,
 W_q , W_v matrices

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Multi-Head Attention

Let's say Q – is the set of states we look **from**

K - is the set of states we look **at**

V - is the set of states we **take values from** (usually $K=V$)

In encoder self-attention $Q=K=V$; *they are represented as matrices (states = rows)*

Each head has its own head-specific parameters, i.e. W_K ,
 W_Q , W_V matrices

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

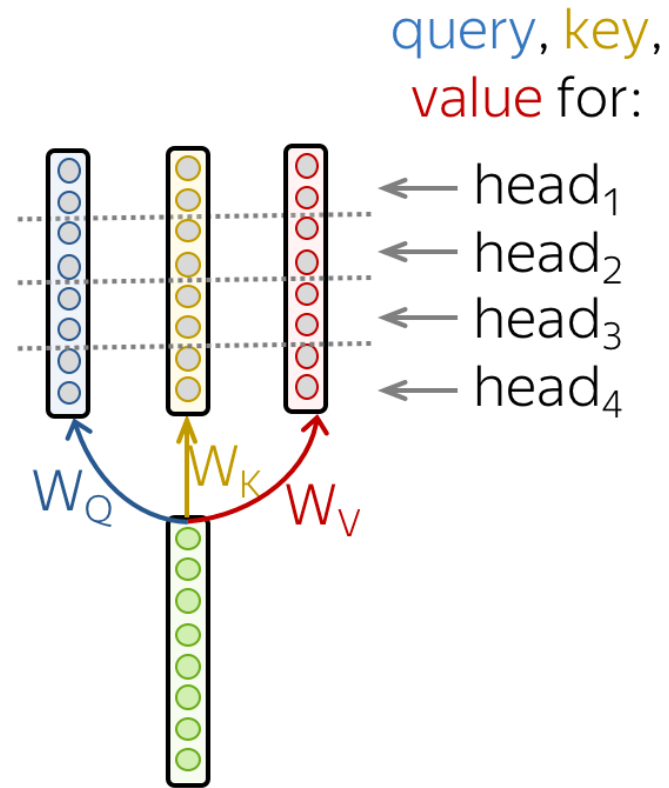
Then, the results are concatenated

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_n)W_o$$

Multi-Head Attention

In practice:

Split equally
into number of
heads parts



Multi-Head Attention

Each head has its own head-specific parameters, i.e. W_k , W_q , W_v matrices

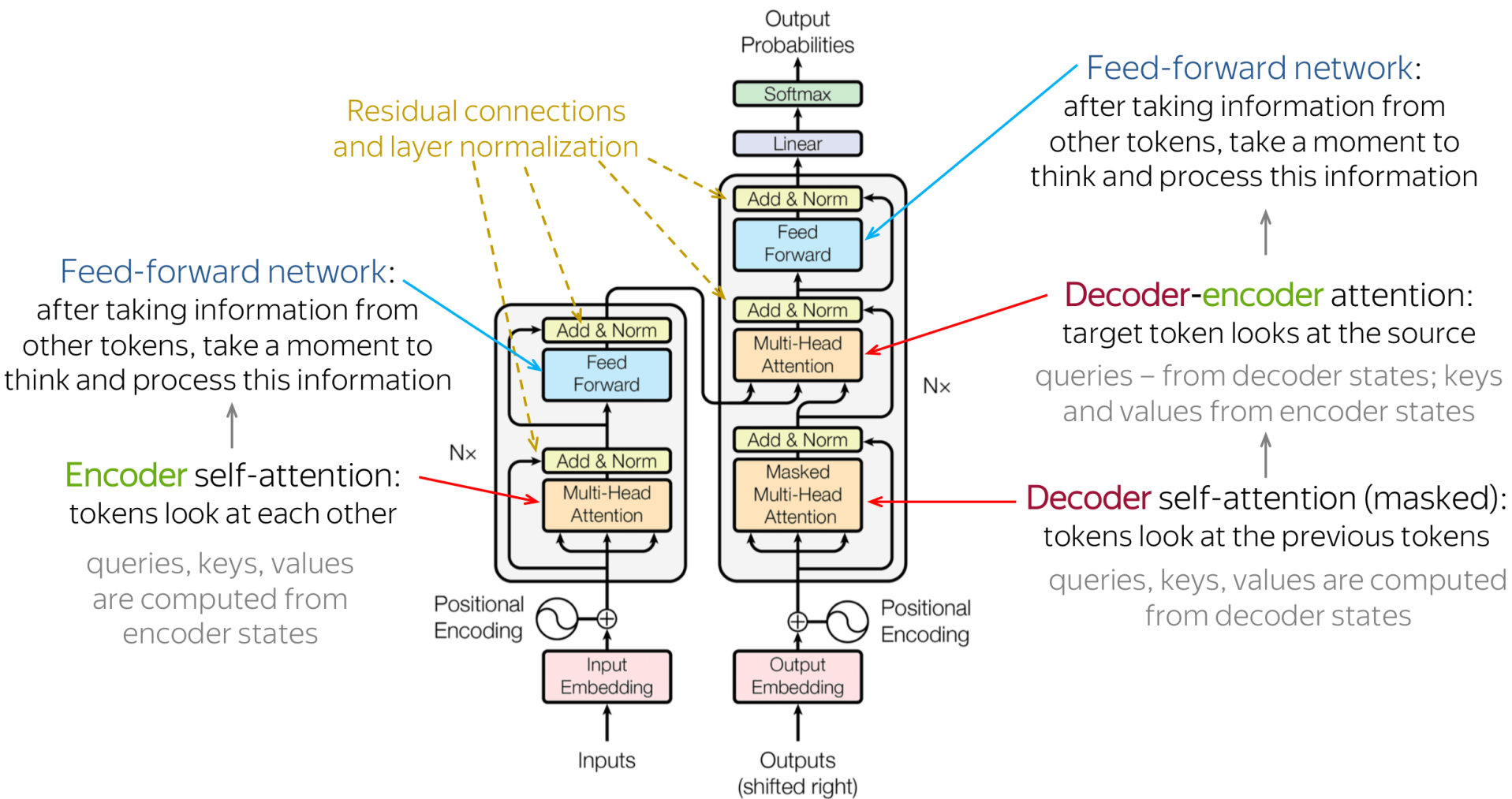
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_n)W_o,$$

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

If we treat softmax scores as ‘constant’ (of course, they are not constant): the result is **a linear function of the token representations**.

Multi-Head Attention only weights and transforms them

Transformer architecture



Take-aways

- Transformer is the architecture which powers most of state-of-the-art models in NLP and beyond
- The key component is multi-head attention

Next time we will continue looking into other Transformer's components

- it will turn out that you're familiar with many ideas